Florian Pudlitz, Florian Brokhausen, Andreas Vogelsang

Extraction of System States from Natural Language Requirements

Pudlitz, F., Brokhausen, F., & Vogelsang, A. (2019). Extraction of System States from Natural Language Requirements. Presented at the 2019 IEEE 27th International Requirements Engineering Conference (RE). https://doi.org/10.1109/re.2019.00031

Extraction of System States from Natural Language Requirements

Florian Pudlitz
Technische Universität Berlin, Germany
florian.pudlitz@tu-berlin.de

Florian Brokhausen
Technische Universität Berlin, Germany
florian.brokhausen@tu-berlin.de

Andreas Vogelsang
Technische Universität Berlin, Germany
andreas.vogelsang@tu-berlin.de

***Abstract*—In recent years, simulations have proven to be an important means to verify the behavior of complex software systems. The different states of a system are monitored in the simulations and are compared against the requirements specification. So far, system states in natural language requirements cannot be automatically linked to signals from the simulation. However, the manual mapping between requirements and simulation is a time-consuming task. Named-entity Recognition is a sub-task from the field of automated information retrieval and is used to classify parts of natural language texts into categories. In this paper, we use a self-trained Named-entity Recognition model with Bidirectional LSTMs and CNNs to extract states from requirements specifications. We present an almost entirely automated approach and an iterative semi-automated approach to train our model. The automated and iterative approach are compared and discussed with respect to the usual manual extraction. We show that the manual extraction of states in 2,000 requirements takes nine hours. Our automated approach achieves an F_1-score of 0.51 with 15 minutes of manual work and the iterative approach achieves an F_1-score of 0.62 with 100 minutes of work.**

***Index Terms*—Natural Language Requirements, Named-entity Recognition, System States, State Extraction**

I. Introduction

Explicit documentation and modeling of system states have several advantages for discussing and analyzing requirements. A number of existing requirement specification techniques address the specification of system states (e.g., State Charts [1]) or reference system states (e.g., as conditions in Use Cases [2]). System states also play a vital role in the verification of system behavior against their requirements. Verification methods such as system tests or simulation often refer to states that are mentioned in the requirements. During requirements verification, a requirements engineer monitors system states while the system is executed. In other approaches, models of system states are used as monitors that continuously check a desirable configuration of system states at runtime [3].

However, in many industrial contexts, requirements are expressed in natural language [4], [5] and system states are not explicitly mentioned. Therefore, requirements engineers who aim at monitoring system states during system executions may profit from tools that suggest or point to potential system states in written requirements. Especially in the automotive domain, the development of vehicle functions is state based. Therefore, states are implicitly included in vehicle specifications and extraction is a particular challenge. Such tools may use linguistic rules to extract state candidates (as, for example, the approach in [6]) or *learn* to identify system states based on annotations in other requirements documents (e.g., by training a Named-entity Recognition model). The performance of learning approaches, however, strongly depends on the amount and quality of labeled data from which the approach can learn.

In this paper, we introduce and compare a semi-automated and an almost entirely automated approach for ML-based Named-entity Recognition to automatically extract system state candidates from natural language requirements. Our focus is on the analysis of requirements from the automotive domain. The two approaches differ in the amount of manual work that needs to be invested by the engineer. Both approaches result in a list of identified system states, which can be used to annotate the set of requirements which these states originate from. In addition, the two approaches provide a trained Named-entity Recognition model that can be used to automatically detect state candidates in new, unseen requirements. In an experimental setting, we compare both approaches with each other in terms of manual work necessary and achieved performance in comparison to an entirely manual state identification. The hypothesis to be examined with this experimental setup is the benefit of automation to reduce otherwise time-consuming manual work. The two approaches shall examine the benefits of scaling the user interaction for the task of identifying system states. Presumably, the semi-automated approach yields better results over the almost entirely automated one, since more expert knowledge is incorporated into the state extraction by having frequent user feedback.

Our results show that, in comparison to the manual labeling process, the approach with minimal manual labor achieves a performance of 0.51 in terms of F_1 with only $3\,\%$ of manual effort. The alternative approach achieves a performance of 0.62 in terms of F_1 with $18.5\,\%$ of manual effort, verifying the aforementioned hypothesis.

In contrast to existing approaches that use linguistic rules, like [6], [7] and [8], the presented approaches do not require a specific structure or style in which requirements need to be written. We conclude that automated machine-learning approaches can be applied with a reasonable amount of effort and have the potential to be used for automatic identification of system states that may be used in requirements analysis and verification.

This paper is structured as follows: In the next section we give more background information. Section III summarizes

the related work. Our approach is described in more detail in Section IV. The evaluation is described in detail in Section V. Finally, we summarize the work in section VI.

II. Background

The complexity of software has increased steadily in recent years. Application areas such as autonomous driving, smart factories, or digitized medical technology are increasing the expectations for reliable software. Systematic verification of software functions is therefore essential. The complexity of software also increases the complexity of test management. In recent years, simulations that model the behavior of the software have become an important tool in test management. Simulation scenarios can represent complex situations and support prototyping without high hardware costs. A special challenge is the linking of natural language requirements to the test scenarios of the simulations in which the requirements are to be verified. Without this link, a reliable verification is not possible.

For the evaluation of complex simulation runs, system states must be detected and evaluated in a targeted manner. These system states are inherent to the natural language requirements and therefore must be identified. The work in [10] shows the importance of system states for creating mode models.

In [11], a method is presented which, among other things, marks states in requirements. In a subsequent step, these states are connected to specific signals of the simulation. The engineer needs a lot of time to manually extract the states. The presented approaches for extraction are primarily manual and time-consuming.

A. Modes

The IEEE standard 29148 for software requirements specifications (SRS) denotes: "Some systems behave quite differently depending on the mode of operation. For example, a control system may have different sets of features depending on its mode: training, normal, or emergency" [12]. The authors propose structuring options depending on the modes to better specify the behavior of the software. In natural language requirements, the idea of specific states helps to clearly define the behavior of the software. The sum of all states formulated within a requirement specification can be described as a mode model [10]. It does not matter whether they are explicitly or implicitly included in the requirements. Requirements that are described on the basis of modes are presented in many papers [13]–[16]. The different understanding and the different use of modes in requirement documents can influence the quality. Especially in [17], it becomes clear that modes are a possible source of misunderstanding and ambiguity in requirement documents. In [10], a mode, i.e. system state, is defined as follows:

> A *mode* is a specific state that describes a system's state of operation. We describe a mode by a name and a set of *mode values* (e.g., `Operation = {Off,Starting,Running}`)

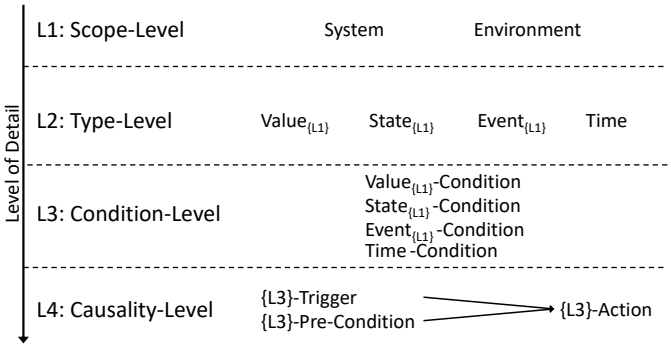

Fig. 1: Overview of Multilevel Markup Language

In their work, they describe different approaches to the detection of these modes. The extraction of modes is a special challenge due to inconsistent naming, implicit use, and differing understanding. The presented extraction possibilities are elicitation by interviews with domain experts, elicitation by feature dependency analysis, and elicitation by requirements inspection. All methods are manual, very time consuming, and hardly possible in the standard development process. Automated detection of system states supports the creation of mode models and improves the understanding as well as the development process.

B. Multilevel Markup Language

The work of [11] describes a multilevel markup language to extract information from natural language requirements. An engineer can annotate text phrases to be observed in a simulation. The annotations can be made at different levels, which depend on one another in a hierarchical manner. Each of the levels signifies a different level of detail on which the requirements can be annotated. Fig. 1 shows the different kinds of annotations and the associated levels.

On level one, text phrases or single words are only distinguished between the two categories System and Environment. On level two it is additionally possible to distinguish between state, event, value, or time information. Based on this, conditions can be created on level three, which are monitored and verified in the simulation. For this, level two annotations are linked to specific state values via a comparison operator. The most complex way to extract information is to create causal relationships at level four. The described approach is based on fully manual annotations made by an engineer. Depending on the time and effort invested by the user, the annotations can be arbitrarily complex.

An automation of level two annotations, which contains states, offers significant benefits to the engineer. It supports the annotation process on level two and upwards. The automated detection of system states is a special challenge. These are rarely associated with numbers or signal names in natural language requirements. This makes an automated identification approach especially challenging. States are defined in the work as follows:

Describing objects with multiple possible, but exclusive states (e.g., door - open/closed).

In complex requirements specifications with more than a thousand single requirements, manual extraction is tedious and time-consuming for the engineer. The manual process is facilitated by the described method but still requires automated support.

III. Related Work

A. Requirements Engineering and Machine Learning

The fact that requirements specifications are mostly written in natural language poses challenges to their refinement due to the inherent complexity of natural language. Therefore, the adaption of natural language processing techniques with machine learning algorithms for requirements engineering is subject to intensive research.

In [18], the authors present a framework facilitated by Artificial Intelligence to support the entire requirements engineering process, from elicitation through quality assurance to the continuous refinement. They apply techniques for natural language processing, ontology reasoning, and deduction.

Another popular field of research is the classification of requirements. A common discrimination is done between functional and non-functional requirements, as reported in [19], and [20]. In [21], the authors present an approach using Convolutional Neural Networks to differentiate between requirements and additional information, which is incorporated in requirements specifications.

B. Requirements and Simulation

Software testing with simulation-support differs between application domains. In recent years, the use of simulations has increased in many areas. Simulations virtually represent the real world with a certain level of detail [22]. They facilitate the virtual testing of all requirements with comparably low hardware costs, since many different test cases can be covered. The verification of requirements with test cases is the subject of intensive research. In [23], the authors show that the research mainly deals with the formalization of requirements in order to subsequently process them automatically. However, linking natural language requirements to simulation runs to assure their verification is not yet in focus. In [24], the authors discuss how requirements of software systems are explored using simulations, but there is no alignment.

Simulations are growing rapidly, especially in the automotive industry. Automated approaches that process natural language requirements with a focus on information for simulations are therefore particularly important.

C. Term Extraction with NER for NL Requirements

The concept of Named-entity Recognition is the detection of certain categories in a text. These categories need to be predefined. Since the CoNLL2003 (Conference on Natural Language Learning) shared task on language independent Named-entity Recognition, there have been many implementations of this method with machine learning algorithms. Conventionally, as predefined in the CoNLL2003 task, entities to be detected are people, organizations, locations and the like. As presented in [25], recent advances towards NER achieve very promising results on the CoNLL2003 task and show significant performance improvements as reported in [26], [27] and [28].

The author in [29] presents an application of NER to the requirements domain. The author aims at automatically creating message sequence charts and automata. Since term consistency is very important, they extract relevant terms like messages with an NER approach. The work in [7] attempts the automated extraction of state machines from natural language requirements. Their extraction of relevant states, however, is done manually. In [8], the authors present an automated transition from use cases to state machines. Their extraction of states is done by explicitly following a set of transformation rules.

The authors in [30] use a NER-like algorithm to automate glossary term extraction for requirements documents.

A particular challenge for the use of NER algorithms is the selection of training data. Bootstrapping approaches were originally used as a method of extracting terms through the recognition of patterns [31]–[33]. In [34] and [35], bootstrapping algorithms are used to automatically label unlabeled data. This data is then used for an NER. The inaccuracy of the bootstrapper, however, reduces the quality of the NER algorithm. In our approach, we involve the user in the training process with minimal time to significantly improve the quality of the NER results. The authors in [36] are concerned about the adaptability of this method to a specific domain. The application of this 2-step approach to the biomedical-domain is presented in [37].

IV. Approach

In this paper, we present two approaches for the extraction of systems states, with differing degrees of automation and user interaction. Fig. 2 and Fig. 3 schematically visualize the semi-automated and the almost automated approach, respectively. As starting point for either approach, a small set of seed states is needed. This set is created by manually scanning a small, random subset of the requirements for contained states. The result is then used as a seed for further processing.

The first approach is almost entirely automated and requires minimal user interaction. For readability reasons, the approach will be referred to as automated for the remainder of this paper. The mentioned list of seed states is used to initialize a bootstrapper. This bootstrapper then detects more states in the training data. The resulting list of states is reviewed by the user, deleting all phrases which do not conform to the state definition. The conditioned set of states is then used to produce a training set for the Named-entity Recognition model. All samples in the training set which contain one or more of the states are extracted and assigned to the corresponding labels. This labeled training set therefore consists solely of requirements which contain states and their assigned correct labels per word, signifying if it is a state or not. This data is then used to train the NER model once.

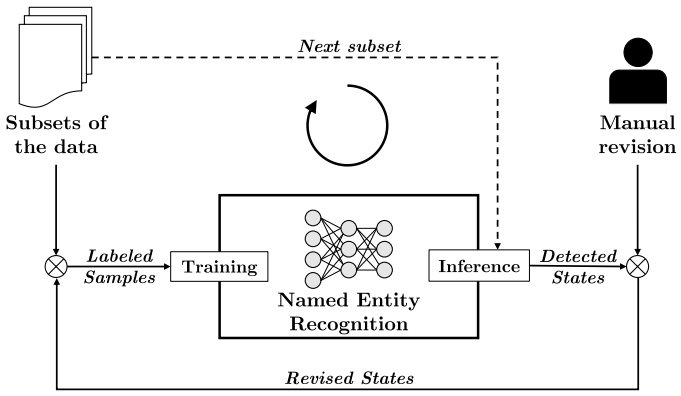

Fig. 2: Schematic overview of the semi-automated approach

The second approach iteratively incorporates the user into the training process. At first, the training data is split into several parts. With the seed states as mentioned before, the relevant samples from the first training subset are extracted and used for the training of the NER model. The trained model is then applied to detect states in the second subset of the training data. The detected states are presented to the user for revision. The reduced, reworked list of detected states is then used to produce a labeled training set of the combination of the first and second subsets of the training data. This procedure is repeated until the number of subsets is exhausted.

A. Definition of States

System states describe the changeable behavior of a system. Depending on the point of view, domain, or system, these states may be defined differently. The definition of a state may further depend on its specific usage. With the focus on software and the distinction between run-time and compile-time, the question arises what parts of a system can change and when. Parameters, features, and design decisions are changeable before each run but remain constant at run-time. In our work, we define states in the context of the automotive industry. Especially the increasing amount of software in the vehicles and the consequential increase in the complexity of requirements documents makes a manual detection of states very difficult. We formally define a `State` as a tuple:

$$State := (Name_x \rightarrow Value_{xy})$$

Accordingly, a state is always assigned to a specific value, which must be explicitly stated in the requirements. The sole occurrence of the name is therefore not sufficient to fulfill the defined requirement of a state. However, if at some point in the requirements specification, a state name is associated with specific values, every occurrence of this state name in the specification is recognized and labeled as such. The corresponding values always describe definitive characteristics and exhibit no continuous progression. They describe changes that are observable in the test process. In our context, the definition includes both software and vehicle states. Table I lists two exemplary requirements including states and non-states, along with a short explanation.

TABLE I: Examples of the definition of states

Requirement	Tuple	State
The button starts and stops the engine.	$Name_1 = engine$ $Value_{1,1} = starts$ $Value_{1,2} = stops$	Yes
	$Name_2 = button$ $Value_{2,1} = \{\}$	No, concrete values are missing
At speeds above 150 mph, the system will be turned off.	$Name_3 = speed$ $Value_{3,1} = \{\}$	No, continuous progression
	$Name_4 = system$ $Value_{4,1} = turned\ off$	Yes

The examples exhibit two states ($engine$, $system$) with corresponding values. Conversely, the terms $button$ and $speed$ are not identified as states since they do not conform to the definition. If a specific state is not consistently used, and synonyms for it exist, these are regarded as unique, independent states as well. If there were more requirements in a document in addition to the two examples, every occurrence of the terms $engine$ and $system$ would be marked as a state as well, regardless if corresponding values are mentioned in the respective requirement or not.

Our definition differs from the one in [10] and [11]. While we also expect a specific value mapped to each name to be identified as a state, we do not incorporate this value into our state definition. Further, we define all occurrences of once identified state names in the requirements as states. In [10], only the explicit combination of name and value is considered. Therefore, in [10], the state *engine* in a different requirements, without explicitly mentioned values, would not be identified as a state. The work in [11] does not consider values in their definition of level two (see II-B) as well. The decision whether a term is considered a state is entirely subjective to the engineer. In comparison, our definition is stricter, since we do require assignable values for a state to be identified as such. However, the approach in [11] can benefit from the one presented in this paper.

B. Data Preprocessing

The requirements data provided by an automotive industry partner are preprocessed to remove redundancies and improve the overall data quality.

Before the data cleaning process, the requirements are split into their constituent sentences. By using the sentences as samples, the number of samples is increased from $10\,377$ requirements to $14\,000$ sentences. During this process, all sentences consisting of only two words or less are deleted. This amounts to 251 deleted sentences. Additionally, the sentences are tokenized by simply splitting at every white space.

The data further exhibits some peculiarities which are handled in the preprocessing. There are multiple symbolical special characters used throughout the requirements like percentage signs or ampersands. These are replaced with either a special

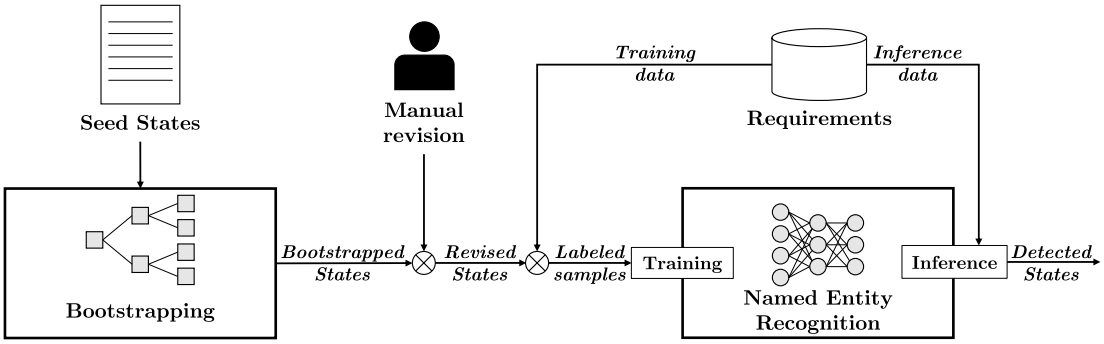

Fig. 3: Schematic overview of the almost automated approach

TABLE II: Quantities of removed symbols and elements

Symbol	Occurrences	Element	Occurrences
Arrow	34	Reference	549
Forward slash	2062	Mail addresse	6
Ampersand	112	Identifier	715
Percentage	762	Enum start	820
Hyperlink	24	Enum item	3356
Number	6595		

token or a sensible wording to simplify downstream processing and not lose semantic information when removing remaining special characters later. Furthermore, there are some frequently occurring elements, e.g., mail addresses, which are replaced by special tokens as well. Other common elements in the requirements domain include references to other documents or chapters as well as special system-specific identifiers for signals or the like. The data further exhibits many enumerations. These are tagged with a special start token and another in front of every element of the enumeration. Additionally, all numbers are removed from the data and replaced with a special token. Table II shows the replaced symbols and elements together with the number of replacements made in the dataset. After this processing, all remaining $33\,138$ special characters are removed from the data to avoid encoding issues in later processing.

The preprocessed data is subsequently used to train the word embedding vectors via the word2vec approach by [38]. To restrict the variance of the data, only words occurring more than two times in the dataset are incorporated into the vocabulary. This amounts to 5455 words in the vocabulary. The embedding size is set to 50 since [28] reports the best results with this setting and, additionally, the vocabulary size of the data used in this work is comparably small. The embedding vectors are trained on the entire dataset. Furthermore, a character embedding is produced. All 32 unique character are included into the set. The character embedding values are initialized according to a random uniform distribution as suggested by [28] and are not pretrained. The embedding size is set to 25.

C. Bootstrapping Algorithm

For many applications that incorporate model training, a large number of training samples is necessary for adequate performance. For special applications like the one in this paper, training data has to be generated first. This process can be automated with bootstrapping algorithms. With bootstrapping, a large training set can be assembled with a small set of starting samples. The resulting labeled data can be used as input for further processing. This paper describes two ways to automate the detection of states. For our automated approach, new states are systematically detected in the data.

The presented approach in [39] extracts entries by learning patterns in texts. In our approach, we apply the presented algorithm to requirements from the automotive context. We use the algorithm to automatically detect states using patterns. The resulting list of states is used as input for the following Named-entity Recognition. The starting point, i.e. seed, is a small selection of states extracted by manually analyzing randomly selected requirements. For the detection of states in all requirements with the bootstrapping algorithm, the extraction of these seed states is the only manual step. Afterward, the algorithm cycles through the following three steps in a completely automated way:

Labeling data and creating patterns: With the selected entries, all requirements samples are labeled. Each matching requirement is examined to recognize patterns. To recognize a pattern, the context around the tagged word is examined in a window of two to four words before and after the target word, for example: *the status of the **X*** or *the signal describing the **X***; where **X** is the examined state candidate.

Scoring Patterns: The recognized patterns are scored. The top scoring ones are incorporated into the set of learned patterns.

Learning entities: With the newly learned patterns, new candidates are identified in the texts and are then evaluated. A scorer ranks the candidates and adds the best to the list.

These three steps are repeated iteratively and new entries and patterns are recognized in each step. The algorithm either terminates when no new patterns are detected or after a specified number of iterations. In our application, the bootstrapping approach automatically detects states in the requirements data without time-consuming user interaction. The engineer just has to create a small seed as a starting point. In order to significantly increase the quality of the data, we integrate a domain expert who assesses the detected states of the bootstrapper once. The exact assessment procedure is explained more detailed in Section IV-E.

D. Named-entity Recognition Model

The Named-entity Recognition model used in this work is implemented according to the one presented in [28]. The general architecture is displayed in Fig. 4.

They incorporate both word and character embeddings into the representation of the input. The word embedding is pre-trained according to the skip-gram model as presented by [38]. Since the automotive requirements domain has a specialized set of terms and phrases, the embedding is trained on the requirements themselves. Additionally, the embedding is adapted during the training of the NER model.

The character embedding is initialized with random vectors for each character and is only adapted during the NER model training. To reduce the multiple character embeddings belonging to a word, a Convolutional Neural Network (CNN) is used. For this, every word is either padded or truncated to a fixed number of characters, so that the input to the network has a consistent size. With a convolutional and a pooling layer, this matrix of character embeddings is reduced to a single vector corresponding to the words character composition.

The two input embeddings are then concatenated for each word to produce a sequence of word vectors for each input requirement.

The actual Named-entity Recognition model is implemented with a Bidirectional LSTM (BiLSTM). This network layer processes the input sequence in a forward and backward fashion. This computation incorporates the sequential nature of the sentences and accounts for the inherent semantic connections between the words. Additionally, due to forward and backward processing, the connections between words are accounted for with respect to the preceding and subsequent context [40]. The forward and backward computations then each produce a new sequence of the same length as the input. For forward and backward processing this output sequence is then fed to a linear fully-connected layer to produce a tuple per word in the input sequence, representing the corresponding category of the word. In this application, this output is a tuple since the model only needs to differentiate between states and non-states. In order to produce labels between 0 and 1, a softmax is applied to the tuples of the sequence. Lastly, the two forward and backward processed label tuples are combined by averaging their respective values. This last layer deviates from the model presented in [28] since for the application in this work, it produced better results. With this last computation, both the forward and backward processing of the input are weighted the same and combined to produce an output between 0 and 1. The individual outcomes of the forward and backward computation are not analyzed.

The output of the network is a tuple, where the first element corresponds to the word being a state and the second value to it not being a state. A model predicting just one value, where a 1 signifies the state-label and a 0 the opposite, performed significantly worse.

In conclusion, our model differs from the one in [28] mainly with respect to the final layer. Additionally, we do not incorporate meta-information about words and characters into our embeddings, as the authors proposed.

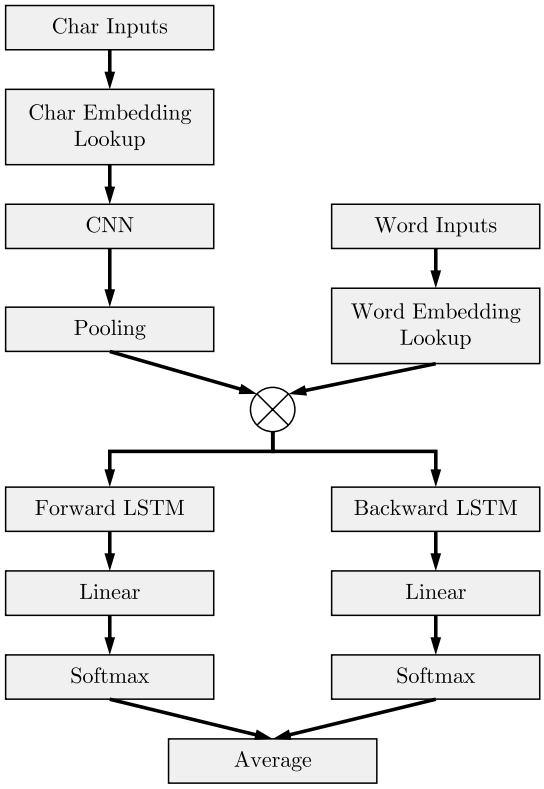

Fig. 4: Layer structure of the Named-entity Recognition model

E. User Interaction

In our paper, we show two approaches aim to support states extraction. The different possibilities for generating the training data require a different time investment of the user. The effort and quality of the models are then compared to purely manual extraction.

Our automated approach requires the least time. During the training process, the user has to revise the suggested list of the bootstrapping algorithm. The user assesses the individual entries as to whether they meet the definition in Section IV-A. Terms that are not states are deleted from the list, while actual states remain in the list unchanged. Since only the name of the state appears at this point, domain knowledge is required to assess whether different system states exist. This minimal effort is required only once during the entire training process. After training the NER model with the revised states of the bootstrapper, states can be recognized in new data without additional time investment.

Our presented semi-automated approach starts with a small set of requirements and extracted candidates. The user now has to revise the candidate list exactly as in the first approach. Entries that do not correspond to the definition are deleted, actual states remain in the list unchanged. Then, a new partition of data is labeled according to the revised states. The NER is retrained and applied to another new set of data. The output is again a list of state candidates. The cycle starts again. The

user is involved here in every cycle. The user's invested time decreases with each cycle as the NER model steadily improves. In our evaluation, we show the qualitative influence of the invested time.

F. Limitations

In our approach, we involve the user as little as possible to significantly improve the quality of our training. In both training methods, the user has a strong influence on the final result. Our focus lies on requirements documents from the automotive sector and requires domain knowledge to reliably candidate quality. This introduces the risk that a poorly trained model may not be very satisfactory in practical use. We see three potential weak spots:

- **Incorrect assessment:** The review of the proposed candidates is subjective to the reviewer and may be affected by different views on the system. We provide a clear definition of states, yet it is possible that states are identified differently depending on the user.
- **Insufficient domain knowledge:** Since states of a system are partly dependent on the implementation, understanding of the system is necessary to properly handle the candidate list.
- **Systematic, accidental error:** The user who is involved in the training process can make mistakes due to fatigue or inattention. Therefore, it is possible that states are erroneously deleted and vice versa.

To avoid human error, we recommend involving several people in the training process.

Another limitation of the presented approaches are the inherent inconsistencies within requirements documents. If there are synonymously used terms for states within the documents, these are not necessarily detected. The approaches do not have the capability to automatically connect synonyms which are used to describe a state. Instead, each synonym for a state is regarded as its own individual state.

V. Evaluation

A. Strategy

In order to properly test and evaluate the two approaches as presented in Section IV, the following strategy is pursued.

The extraction of seed states is done on 50 randomly selected requirements. This manual extraction yielded eleven seed states. This scope was chosen to keep the initial work put into the two approaches as minimal as possible, while still yielding sufficient input for a proper execution.

The preprocessed $14\,000$ samples in the dataset are randomly shuffled under the assumption, that the requirements specification which the sample originated from does not affect the subsequent processing. The model should be able to handle the training and inference tasks regardless of samples' origin. Additionally, all specifications are from the automotive domain which further supports the independency claim.

The training, validation and test split are conducted as follows. With the assumptions mentioned above, the shuffled data can be split in any manner to produce a viable subset.

TABLE III: Parameter setting of the NER model

Parameter	Value	Parameter	Value
LSTM size	275	Optimizer	Adam
LSTM dropout	0.68	Epochs	50
Maximum sentence length	50	Batch size	5
Maximum word length	15	Learning rate	0.002
Detection threshold	0.3		

2000 of the randomly shuffled samples are split off to serve as test data to evaluate model performance. For the automated approach, the remaining $12\,000$ samples are used to train the NER model. During training, $10\,\%$ of the data serve as validation data. For the semi-automated approach, the $12\,000$ samples are randomly split into 10 disjoint subsets to enable the execution of training iterations. Within each training iteration, the trailing $10\,\%$ of data are used as validation data.

The test data is additionally annotated to serve as the gold standard, which all models are evaluated with. Hence, all states included in the 2000 samples need to be identified. In order to achieve this, four experts from the automotive sector analyze the data. The data is disjointly split between the reviewers. After each expert extracted the relevant states from the respective subset, the resulting list is cross-validated. For this, every expert reviews the state list of another one. In this second round, the expert is only allowed to remove states from the list; no new states are introduced. Therefore, the resulting gold standard list only contains states which two of the experts agree upon.

The list of gold standard states facilitates the labeling of the test data according to the NER model output, assigning each occurrence of a states the respective correct label. The manual extraction of states yields 222 unique states in the test set. Of the 2000 samples in the test set, 1228 samples actually contain a state.

B. Model Implementation

The Named-entity Recognition model has several hyperparameters, displayed in Table III. These are kept constant over the course of the training iterations as well as for the automated approach.

The number of epochs is to be interpreted as the maximum number of epochs. During training, the model is augmented with early stopping, which means that the training is terminated if the validation loss has not improved over the course of the last 5 epochs. The learning rate is determined empirically. Except of the batch size, the remaining parameters are set according to the configuration in [28]. The batch size is chosen to be smaller than suggested, since there are fewer training samples as in comparable datasets - especially during early iterations.

The parameters for maximum sentence and word lengths are determined to incorporate as much information as possible while still supporting reasonable computation. To determine a sensible limit, the histograms of sample and word lengths in Fig. 5 are analyzed. By cutting longer samples in accordance with the specified maximum length, $93.94\,\%$ of the data is used. The maximum word length restricts the number of

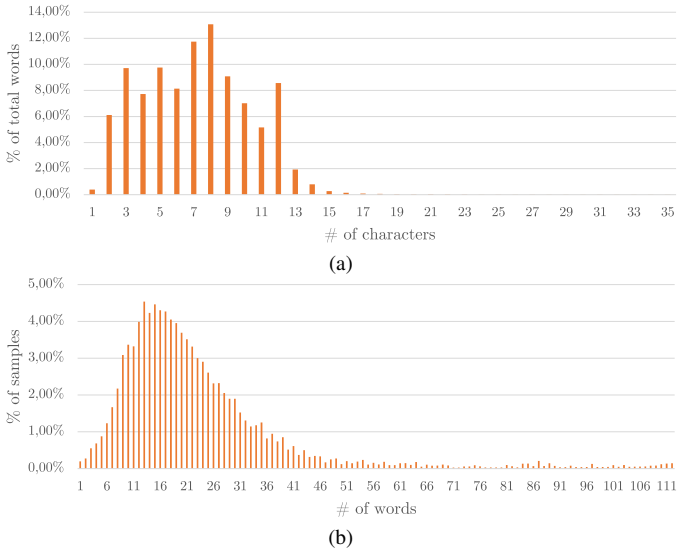

Fig. 5: Histograms of the number of characters per word (a) and number of words per sample (b)

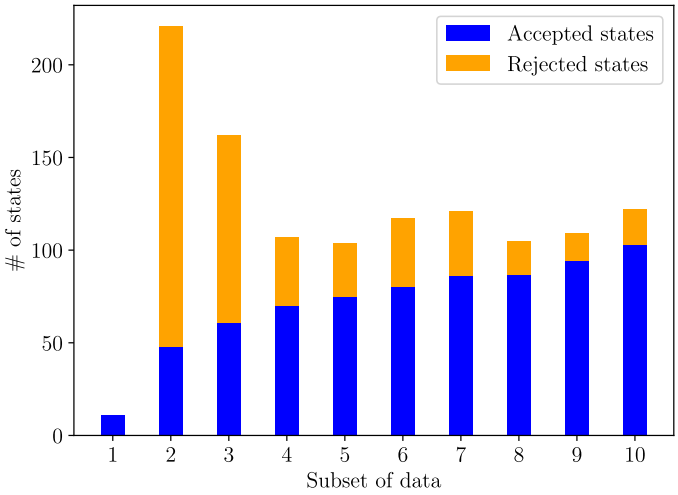

Fig. 6: Detected states per iteration

characters being incorporated by the character encoding CNN. The specified value accounts for 99.48 % of all characters in the data.

Lastly, the detection threshold defines the limit, above which the model output is interpreted as signifying a state. The value is set empirically by observing precision and recall performance on the test set for alternating values.

C. Dataset Specifics

Our approach focuses on the extraction of system states. Section II discusses the importance of these states in the automotive sector. Our data for both the training and test sets are provided by a large German automotive group. The company differentiates between component specifications and system specifications. The component specification describes individual small vehicle components such as sensors or motors, each of which covers a specific functionality. In system specifications, several components are combined into a larger, cross-component vehicle function. A special characteristic of automotive requirements is the accumulation of signal and function names. A typical requirement with signal names is, for example: “When the component BSM receives the signal BSM_Stat_Req, the system state changes to ON”. These are not consistently used and are, therefore, particularly difficult to process automatically.

For our approach, we included both types of specifications. All requirements are written in the English language. The requirements were created manually by requirements experts of the company. The requirements are written in prose form and without restrictions, patterns or templates. Therefore, it depends on the writer whether British or American English spelling is used, which is why both cases occur in our data. Despite the review process of the experts, some syntactical and spelling errors are still observed in the data.

D. Results

The results follow the strategy presented in Section V-A.

Fig. 6 shows the number of detected states for each subset of data as well as the ratio of rejected and accepted states. In the first iteration, only the eleven seed states are displayed, since the first subset of the data on its own only serves as training data. As the model is trained with increasing numbers of training samples over the course of the iterations, the number of predicted states decreases. After the third iteration, however, this number stagnates and fluctuates around values slightly above 100. Nevertheless, the amount of correct states with regard to the entire set of predictions steadily increases. At every iteration, the model is able to identify more new, valid states within the data.

This correlates reasonably with the training set sizes displayed in Fig. 7. At each iteration, the theoretically available training data increases by 1200 samples as a new subset is included. Fig. 7 also shows the share of data actually containing the accepted states of the iteration, which actually serves as training data. The ratio of training data to available data stays relatively constant, except for the first iteration. There, only the 11 seed states serve as input to extract relevant samples. Therefore, only about 10 % of the data that actually do contain these states is used for training. For iteration two through seven, the training samples account for about 37 % of the data. In the last three iterations this ratio increases to around 43 %.

The semi-automated approach is further validated on the gold standard. After each iteration, the model is applied to the test data and the results are compared to the gold standard. Fig. 8 shows the precision, recall and F_1 metrics, which are calculated in two different ways. Fig. 8a reports the metrics based on how many of the actual labels in all of the samples the model predicted correctly. The reported scores are the average scores of all samples in the test set. This accounts for multiple occurrences of states in the data, since the model should be able to identify a state in every sentence it occurs reliably. Additionally, this metric also incorporates partial matches of states. If a state is constituted of multiple words, the output

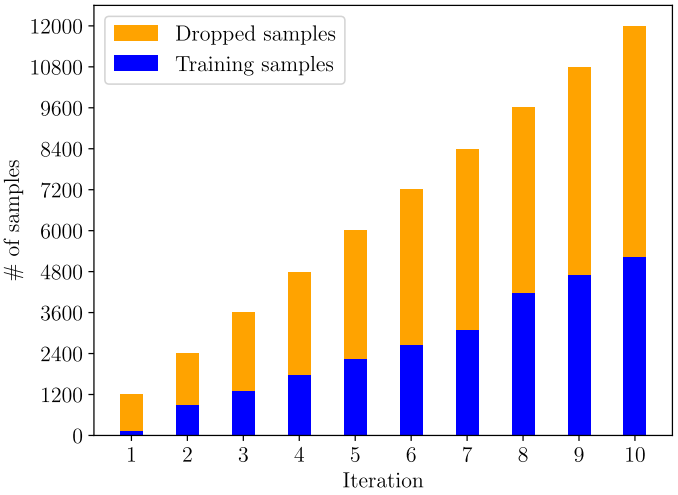

Fig. 7: Training samples per iteration

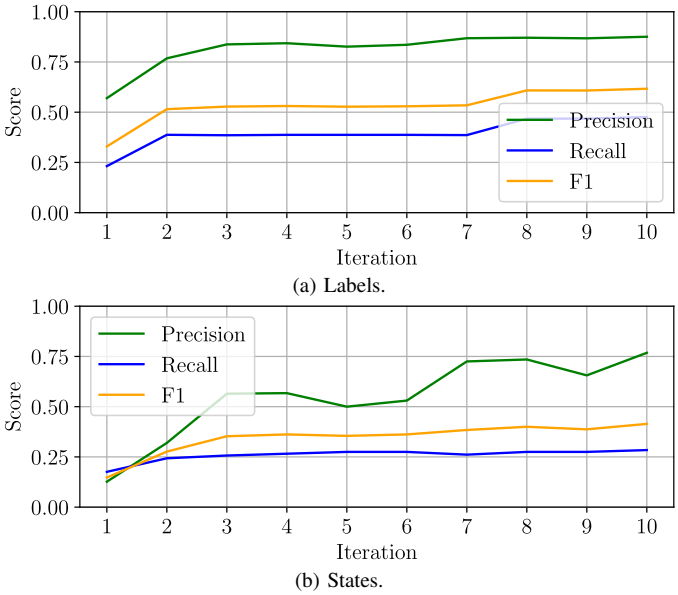

Fig. 8: Performance of the semi-automated approach on the test data

of the model is also relevant if it only labels some of the constituent words correctly.

The graph shows that the model has two distinct phases of improvement, one in the beginning and one in towards the last iterations. The achieved precision of 0.88 signifies the confidence of the model, meaning that words labeled as states are most likely correctly labeled so. The recall on the other hand accounts for the share of correctly labeled states with respect to all actually correct labels. With a recall of 0.48, the model predicts about half of the state labels in the dataset.

The second performance evaluation is displayed in Fig. 8b. Here, the metrics regard the actual states that the model detects, compared to the gold list of actual states in the test set. The precision almost continuously improves over the course of the iterations, while the recall stagnates and only improves by a small amount. This evaluation shows the same pattern as the previous one. The precision achieves a final value of 0.77, again signifying the models confidence in the predicted states. Put into perspective, this means that three quarters of the predicted states are correct. However, the recall examines a weaker performance once again, with a final value of 0.28. Therefore, the model detects just above a quarter of the states contained in the test set.

The automated approach has the same database as the semi-automated one. The bootstrapper receives the eleven seed states as an input to detect more states in the 12 000 training samples. After 20 iterations, the algorithms finds a total of 200 states. After manual revision, 73 states are identified as valid and serve as input to the NER model training. When extracting the relevant samples based on these states, 4472 of the 12 000 samples remain in the training set.

The automated approach does not perform as well as the semi-automated approach. Table IV gives an overview of the achieved scores for both approaches. When comparing the two approaches based on the metrics measured on the label prediction, as explained above, the automated approach performs slightly worse. The performance deviates circa 0.1 from the semi-automated one for all metrics. Therefore,

TABLE IV: Best performance metrics for both approaches

	Metric	Automated	Semi-automated
Labels	Precision	0.77	0.88
	Recall	0.38	0.48
	F_1	0.51	0.62
States	Precision	0.44	0.77
	Recall	0.30	0.28
	F_1	0.36	0.41

when considering the ability of the approach to label words correctly in a given document, the automated approach exhibits comparable capabilities to the semi-automated one.

The second part of Table IV describes the performance on the actually detected states, as mentioned above. Regarding the precision, the automated approach shows significant deviations from the semi-automated one. The model trained with the automated approach is not as confident in the predictions. The recall on the other hand, is even slightly better. With the automated approach, about a third of all states in the test set are detected.

E. Threats to validity

The evaluation shows the comparison of our two approaches to (semi-)automatically extract states from natural language requirements. The evaluation results are subjective to some critical influences. In the following, we want to address three identified threats to the validity of our evaluation and our efforts to minimize those influences.

All results refer to the identified states of the gold standard, which poses a threat to construct validity. Since there is no public data of states from requirements specifications, a gold standard had to be created manually. This process is error prone and directly affects the evaluation results. To reduce the possible misjudgments of an individual expert, four experts

were involved. The gold standard was created in two processing steps, one for the extraction of states and one for the review of the identified candidates. Section V-A describes the exact distribution of the data in more detail.

The chosen hyper-parameters of our approach also influence the results and are therefore an internal treat to validity. The chosen parameters are based on state-of-the-art research. An investigation of the results with respect to the parameter space was not conducted, as we did not focus on parameter optimization in this paper. In the future, a parameter optimization specific to the problem and the available data could further improve the results.

An additional external threat is the involvement of the user in our approach. While this involvement does improve the results for the semi-automated approach, it also directly influences the evaluation. Especially the elicitation of seed states has a significant impact on the performance of both approaches. As described in Section V-A, user interaction is purposely limited. To further minimize this influence, two experts were involved in all necessary user interactions for the evaluation. The experts discussed the decisions taking into account the necessary domain knowledge.

F. Discussion

Both presented approaches incorporate the user into the training process. The shared starting point is a seed of 11 states. These were extracted by reading 50 randomly sampled requirements. In our evaluation this elicitation takes 17 minutes. In practice, it is conceivable that engineers with domain knowledge create this seed without having to read a set of requirements. In this case, the effort is negligible.

In the evaluation, we show results of both approaches, each with a different time investment of the user. The least time is needed in the automated approach. The user has to read and assess the proposed candidates of the bootstrapper. In our evaluation, it takes 15 minutes to scan the list and sort out all terms that do not signify states. All further steps are fully automated. Applied to the gold standard, a precision of 0.77 and a recall of 0.38 are achieved. In contrast, the semi-automated approach achieves a precision of 0.88 and a recall of 0.48. Here the user is involved more in the training process. In each iteration, the candidates have to be revised. In our scenario, the user spent a total of 100 minutes for the revision of states in 10 iterations. At this point, it is important to mention that the training process only takes place once and the created model can be applied repeatedly to extract states in new data. Until now, there is only manual extraction to compare our approaches against. For reading 2000 requirements and detecting the states, our group of experts needs nine hours. On the one hand, with manual extraction, it is sure to have identified a large majority of states, if not all. On the other hand, a new set of 2000 requirements will require another nine hours of work, whereas our trained model can extract states in new data in an instant.

When comparing the two approaches with regard to the detected states, some significant differences can be observed. The automatic approach is set up according to the state-of-the-art when it comes to information extraction [32], [41], [42]. This approach examines a high number of false positives. Some of these candidates do seem reasonable to be identified as states, but are out of scope for a simulation of the system. These include the names and IDs of other systems or also static properties of the system like the Automotive Safety Integrity Level (ASIL) [43], which has defined state values and could therefore be characterized as a state. However, there also are a lot of futile candidates like simple verbs as well.

The semi automated approach is more refined and produces about 88 % less false positives, some of which are sensibly characterized but, as stated before, are out of scope. This improved precision is reasonable since the engineer continuously refines the predictions of the model to improve the quality of training data.

As described in Section II, there are several application possibilities for our approach. The proposed states can greatly assist the creation of mode models or the manual annotation of conditions in requirements specifications.

When using the multilevel markup language, high precision is prioritized over recall. Therefore, it is important that recognized states are correct. Our approach can pre-annotate states, and in the best case as tested, 88 % of states are annotated correctly. A central aspect of the markup language is the freedom of the user to make manual annotations. The engineer can decide which of the pre-annotated states are to be observed in the simulation. It is also possible to use pre-annotated specifications without additional user interaction.

Fig. 8b shows that the precision improves as the number of iterations increases. The multilevel markup language is a great way to get more data about states. This information could further improve our approach and be used for further training. By combining the markup language with our approach, the proposals improve over time without explicit time investment for the training process.

VI. Conclusion

Detecting software conditions is especially important for simulation and testing. As starting point for every system, natural language requirements are used - especially in the automotive industry. Extraction of such states from natural language requirements is a challenge for large complex software systems. We present two approaches to automated state extraction. With minimal time investment in the training process, the extraction process is greatly improved. Our evaluation shows the advantage of little time investment in the training process on the results of the automated extraction. With 100 minutes of user interaction, states can be extracted with a precision of 0.88. Experts from requirements management would need about nine hours for the manual extraction in 2000 requirements.

The test processes can be carried out faster and the automated comparison with natural language requirements is facilitated.

In the future, innovative software features may be the focus of research to better cope with the high complexity of today's software. Our results show that with some user effort, the state extraction can be significantly supportive for the user.

REFERENCES

[1] D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, 1987.

[2] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley Professional, 2001.

[3] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[4] L. Mich, F. Mariangela, and I. Pierluigi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering*, vol. 9, no. 1, 2004.

[5] M. Kassab, C. Neill, and P. Laplante, "State of practice in requirements engineering: contemporary data," *Innovations in Systems and Software Engineering*, vol. 10, no. 4, pp. 235–241, 2014. [Online]. Available: https://doi.org/10.1007/s11334-014-0232-4

[6] L. Kof and B. Penzenstadler, "From requirements to models: Feedback generation as a result of formalization," in *Advanced Information Systems Engineering*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 6741.

[7] B. Walter, J. Martin, J. Schmidt, H. Dettki, and S. Rudolph, "Executable state machines derived from structured textual requirements - connecting requirements and formal system design," in *7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2019, pp. 195–202.

[8] T. Yue, S. Ali, and L. Briand, "Automated transition from use cases to UML state machines to support state-based testing," in *Modelling Foundations and Applications*, R. B. France, J. M. Kuester, B. Bordbar, and R. F. Paige, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 115–131.

[9] P. Rook, "Controlling software projects," *Software Engineering Journal*, vol. 1, no. 1, pp. 7–16, 1986.

[10] A. Vogelsang, H. Femmer, and C. Winkler, "Systematic elicitation of mode models for multifunctional systems," in *23rd IEEE International Requirements Engineering Conference (RE)*, 2015.

[11] F. Pudlitz, A. Vogelsang, and F. Brokhausen, "A lightweight multilevel markup language for connecting software requirements and simulations," in *Requirements Engineering: Foundation for Software Quality*, E. Knauss and M. Goedicke, Eds. Cham: Springer International Publishing, 2019, pp. 151–166.

[12] IEEE, "Systems and software engineering – life cycle processes – requirements engineering," *ISO/IEC/IEEE 29148:2011(E)*, 2011.

[13] M. Broy, "Multifunctional software systems: Structured modeling and specification of functional requirements," *Science of Computer Programming*, vol. 75, no. 12, pp. 1193–1214, 2010.

[14] D. Dietrich and J. M. Atlee, "A mode-based pattern for feature requirements, and a generic feature interface," in *2013 21st IEEE International Requirements Engineering Conference (RE)*, 2013, pp. 82–91.

[15] C. Heitmeyer, J. Kirby, and B. Labaw, "The scr method for formally specifying, verifying, and validating requirements: Tool support," in *Proceedings of the 19th International Conference on Software Engineering*, ser. ICSE '97. New York, NY, USA: ACM, 1997, pp. 610–611.

[16] P. Shaker, J. M. Atlee, and S. Wang, "A feature-oriented requirements modelling language," in *2012 20th IEEE International Requirements Engineering Conference (RE)*, 2012, pp. 151–160.

[17] A. Vogelsang, H. Femmer, and C. Winkler, "Take care of your modes! an investigation of defects in automotive requirements," in *22nd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, M. Daneva and O. Pastor, Eds. Springer, 2016, pp. 161–167.

[18] S. J. Körner, M. Landhäußer, and W. F. Tichy, "Transferring research into the real world: How to improve re with ai in the automotive industry," in *2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*. IEEE, 2014, pp. 13–18.

[19] A. Casamayor, D. Godoy, and M. Campo, "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach," *Information and Software Technology*, vol. 52, no. 4, pp. 436–445, 2010.

[20] A. Dekhtyar and V. Fong, "Re data challenge: Requirements identification with word2vec and tensorflow," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 2017, pp. 484–489.

[21] J. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2016, pp. 39–45.

[22] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*. Prentice Hall, 2000.

[23] Z. A. Barmi, A. H. Ebrahimi, and R. Feldt, "Alignment of requirements specification and testing: A systematic mapping study," in *IEEE International Conference on Software Testing, Verification and Validation Workshops*, 2011.

[24] A. Gregoriades, M. Pampaka, and A. Sutcliffe, "Simulation-based requirements discovery for smart driver assistive technologies," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, 2014, pp. 317–318.

[25] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," 2017.

[26] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," 2017.

[27] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," *arXiv:1603.01360*, 2016.

[28] J. P. Chiu and E. Nichols, "Named entity recognition with bidirectional lstm-cnns," *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 357–370, 2016. [Online]. Available: https://transacl.org/ojs/index.php/tacl/article/view/792

[29] L. Kof, "Requirements analysis: concept extraction and translation of textual specifications to executable models," in *International Conference on Application of Natural Language to Information Systems*. Springer, 2009, pp. 79–90.

[30] A. Dwarakanath, R. R. Ramnani, and S. Sengupta, "Automatic extraction of glossary terms from natural language requirements," in *2013 21st IEEE International Requirements Engineering Conference (RE)*. IEEE, 2013, pp. 314–319.

[31] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *Proceedings of the 14th Conference on Computational Linguistics - Volume 2*, ser. COLING '92. Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, pp. 539–545. [Online]. Available: https://doi.org/10.3115/992133.992154

[32] E. Riloff, "Automatically generating extraction patterns from untagged text," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, ser. AAAI'96. AAAI Press, 1996, pp. 1044–1049. [Online]. Available: http://dl.acm.org/citation.cfm?id=1864519.1864542

[33] M. Thelen and E. Riloff, "A bootstrapping method for learning semantic lexicons using extraction pattern contexts," in *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, ser. EMNLP '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 214–221. [Online]. Available: https://doi.org/10.3115/1118693.1118721

[34] Z. Kozareva, "Bootstrapping named entity recognition with automatically generated gazetteer lists," in *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, ser. EACL '06. Stroudsburg, PA, USA: Association for Computational Linguistics, 2006, pp. 15–21.

[35] J. Teixeira, L. Sarmento, and E. Oliveira, "A bootstrapping approach for training a ner with conditional random fields," in *Progress in Artificial Intelligence*, L. Antunes and H. S. Pinto, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 664–678.

[36] D. Wu, W. S. Lee, N. Ye, and H. L. Chieu, "Domain adaptive bootstrapping for named entity recognition," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3 - Volume 3*, ser. EMNLP '09. Stroudsburg, PA, USA: Association for Computational Linguistics, 2009, pp. 1523–1532.

[37] A. Vlachos and C. Gasperin, "Bootstrapping and evaluating named entity recognition in the biomedical domain," in *Proceedings of the HLT-NAACL BioNLP Workshop on Linking Natural Language and Biology*, ser. LNLBioNLP '06. Stroudsburg, PA, USA: Association for Computational Linguistics, 2006, pp. 138–145.

[38] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[39] S. Gupta and C. D. Manning, "Improved pattern learning for bootstrapped entity extraction," in *Computational Natural Language Learning (CoNLL)*, 2014.

[40] A. Graves, A. rahman Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

[41] E. Riloff, J. Wiebe, and T. Wilson, "Learning subjective nouns using extraction pattern bootstrapping," in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, ser. CONLL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 25–32. [Online]. Available: https://doi.org/10.3115/1119176.1119180

[42] A. Maedche, G. Neumann, and S. Staab, "Bootstrapping an ontology-based information extraction system," 2002.

[43] *ISO/DIS 26262 Road vehicles – Functional safety*, ISO/DIS Draft International Standard Std., 2018.